

CDAT Newsletter, April 2006, Tech Tips

CDAT Newsletter

Vol2, April, 2006

[Home](#) [Archive](#)[CDAT](#)[News](#)[TechTips](#)[Contact](#)

Tech Tips for beginners

Basic querying commands in CDMS.

We will learn all about the file, dimensions, variables and axis, and their attributes by using basic querying commands.

How to run the tutorial: Start python or cdat by typing
'python' or 'cdat' at a shell prompt, then copy and paste the text in blue, the text in red is what you will see as output.

Let's start by opening a file we will query

```
import cdms,sys,cdtime
f=cdms.open(sys.prefix+'/sample_data/clt.nc')
```

1. Query "file"

a) global attributes

```
g =f.listglobal() # returns list of global
attributes
print g
```

The output is:

```
['center', 'comments', 'Conventions', 'model']
```

```
m=f.model # get the value of an attribute
'model'
print m
```

Advanced Tech Tips

VCS template layout.

There seem to be a very common question as to how to make your plot look exactly as you want it.

Through this tip, you can learn how to change layout of your plot, sizing, axis labels, text position, fonts, orientation, label position & colorbar position.

How to run the tutorial: Start python or cdat by typing
'python' or 'cdat' at a shell prompt, then copy and paste the text in blue, the text in red is what you will see as output.

First of all, let us understand what is happening when we plot some data on a vcs canvas. The data is plotted with a default settings on what is being plotted and where (like what text appears, where on the canvas, how much space the plot takes, what are the axis labels and tick marks, where the colorbar appears and how it is labeled). All of that is defined in a default template that is used when plotting a data.

If we want to change the plot layout, we should first create a new object, a template, that we will change to our liking. We can then save this template for future use.

Let's grab some data and plot with a default template

```
import cdms,sys,cdtime,vcs
f=cdms.open(sys.prefix+'/sample_data/clt.nc')
```

```
'TR7.1'
```

b) variables

```
vars=f.listvariable() #return list of variables  
print vars  
['clt','u','v']
```

c) dimensions

```
dim=f.listdimension() #return list of  
dimensions  
print dim  
['plev', 'latitude1', 'latitude2', 'time1',  
'longitude1',  
'longitude2', 'longitude', 'time', 'latitude',  
'plev1', 'time2']
```

```
dim_u=f.listdimension('u') # dimensions of 'u'  
print dim_u  
['time1','plev' 'latitude1', 'latitude2' ]
```

2. Query "variable"

First let's get some variable, for example variable 'v'

```
v = f('v') # transient variable 'v'
```

a) attributes

```
att_v = v.listattributes() #return list of  
variables  
print att_v  
['missing_value', 'name', 'title', 'type', 'time',  
'date',  
'source', 'units']
```

```
v_title = v.title #return variables title  
print v_title
```

Monthly Mean Northward Wind Speed

```
v_units = v.units #return variables units  
print v_units  
m/s
```

b) dimensions

```
# Extract a 3D data set and get a subset of the time  
dimension  
data = f('clt', longitude=(-180, 180), latitude = (-90.,  
90.))  
v = vcs.init() # Initial VCS canvas  
v.plot(data) # plot data with a default template
```

Let's see what templates are available to us

```
v.show('template') # Show the list of persistent  
templates.
```

```
*****Template Names  
List*****  
( 1): ASD           ASD1           ASD10  
( 4): ASD11         ASD12         ASD13  
...  
( 91): por_topof3   por_topof3_dud   quick  
( 94): top_of2  
*****End Template Names  
List*****
```

As you can see there is a lot of persistent templates (94). We will use the standard 'ASD' template as a start template, that we will then modify to suit our needs.

```
# Create a new template from the existing 'ASD'  
template  
t_asd = v.createtemplate( 'new', 'ASD' )  
  
# Plot the data using the above 'ASD' new template.  
v.clear() # clear the canvas first v.plot( data, t_asd )
```

To list template members, use the following commands:

```
t_asd.list()          # list the templates members  
t_asd.list('text')    # list only text members  
t_asd.list('xlabels') # list only xlabel members  
t_asd.list('legend') # list only legend member  
t_asd.list('data')   # list only data member  
t_asd.list('mean')   # list only mean member and its  
values
```

For more complete list see: [Creating and Modifying Templates](#).

Each of the template members have the following attributes (among others):

- priority (value 1/0) – show or suppress a member
- x1,y1 – the position of a member (lower left point)
- x2,y2 – the position – upper right corner (if a 'box' member)

```

v_dims = v.listdimnames() #return list of
dimensions
print v_dims
['time2', 'plev1', 'latitude2', 'longitude2']

s = v.shape # return the shape of this variable
print s
(1, 2, 80, 97)

print v.info() # return all the info about the
variable. It's dimensions, axis,
attributes etc.

```

c) axis

From 'v_dim' we know what axes the variable 'v' has. Let's query these axes.

First axes is a time axis, you can get it with those equivalent commands:

```

t = v.getAxis(0) # return the first variable 'v'
axis
t=v.getTime() # return the time axis
print t

```

id: time2

Designated a time axis.

units: months since 1978-12

Length: 1

First: 1.0

Last: 1.0

Other axis attributes:

calendar: proleptic_gregorian

axis: T

Python id: -0x48f56974

To get the values just type:

```

print t[:] # print values of the time axis
[1,]

```

Similarly you can get the latitude axis this way:

```

lat=v.getLatitude() # return the latitude
print lat

```

id: latitude2

Designated a latitude axis.

units: degrees_north

Length: 80

The position is given by a normalized coordinates [0.,1.] .

Setting priority to 0 will cause the member to not be plotted.

The following plot is showing the names, positions and values of most important member for that template. Click on the plot to see a bigger picture

By default when we change a member attribute, the vcs canvas will get updates, to turn it off set :
v.mode=0 # turn the automatic update off

#For example, if we want to suppress the 'mean', 'min' and 'max' texts, we would set their priority to 0 :

```

t_asd.mean.priority = 0
t_asd.min.priority = 0
t_asd.max.priority = 0

```

```

# to move the x-position of a mean string 0.1 to the
right:
xmean_current = t_asd.mean.x
t_asd.mean.x = xmean_current + 0.1
t_asd.max.priority = 1

```

Let's move the colorbar, so that it is vertical and displayed on the right side of the boxfill plot.

The data member is the area the main plot occupies – the boxfill plot, the box1 member is the outline of this area.

```

# to move the right side of a plot to the left to make
space for the legend
# first move the inner data plot
t_asd.data.x2 = 0.87

```

```

# then move the surrounding box i.e the right y-axis
t_asd.box1.x2 = 0.87

```

```

# set the top x-axis (second y axis) to be blank
t_asd.xlabel2.priority = 0
t_asd.xtic2.priority = 0

```

```

# set the right y-axis (second y axis) to be blank
(priority=0)
t_asd.ylabel2.priority = 0
t_asd.ytic2.priority = 0

```

```

First: -88.2883987427
Last: 88.2883987427
Other axis attributes:
  axis: Y
  Python id: -0x48f568f4

print lat[:4] # print first 4 values

[-88.28839874,-86.07109833,-83.84079742,
 -81.60739899]

print lat.getBounds():4 # print only first 4
bounds

[[-89.39704895,-87.17974854,]
 [-87.17974854,-84.95594788,]
 [-84.95594788,-82.72409821,]
 [-82.72409821,-80.49019623,] ]

```

d) time axis

The time axis is somewhat special, because it holds the time object. As you recall, when we printed the time axis variable 't' (using 'print t[:]') we got a value '1.'. If you also recall the info on time axis (that we got from 'print t' command), you will realize that the value '1.' means 'first month from the fixed start time', which in this case is '1.0 months since 1978-12' i.e. '1979-01-01'.

Of course, you can get this info directly from CDAT :

```

time_comp = t.asComponentTime(): # print
time in a more
user friendly way
print time_comp
[1979-1-1 0:0:0.0]

```

Here is how you can go back and forth from so called 'component time' and a 'relative time'.

A component time is an absolute time with representation (year, month, day, hour, minute, second). A relative time is represented as (value, units since basetime).

The component time is in this case '[1979-1-1 0:0:0.0]' and the relative time is '1.00 months since 1978-12'

```

# move the colorbar legend position, to be vertical and
to the right
t_asd.legend.x1=0.9
t_asd.legend.y1=0.82
t_asd.legend.x2=0.95
t_asd.legend.y2=0.3

```

```

# clear the canvas and plot the template again
v.clear()
v.plot( data, t_asd )

```

Plotting Using Keyword Arguments

The plot function has many overriding keyword arguments that control textural and graphical output of the display. Below is a plot() function showing the uses of array objects, template object, graphics method object, and key word arguments. Objects placed in brackets "[]" indicate optional entries into the plot function:

```

plot(array1, [array2 [, template [, graphics method [, key=value [, key=value [, ...]] ] ] ] ]),

```

where array1 and array2 are Numeric arrays or MA, MV or Python lists; template represents a template object; graphics method represents a graphics method object (such as, boxfill or isofill); and key=value represents one variable attributes. If no template is specified, then the default template will be used. If no graphics method is specified, then the default boxfill graphics method is used.

Here are some more important keywords:

```

comment1 = string # Comment plotted above
file_comment
comment2 = string # Comment plotted above
comment1
comment3 = string # Comment plotted above
comment2
comment4 = string # Comment plotted above
comment3
file_comment = string # Comment (defaults to
file.comment)
long_name = string # Descriptive variable name
name = string # Variable name (defaults to var.id)
time = cdtime # instance (relative or absolute),
units = string # Variable units

```

```

[xyl|zltlw]name = string # x or y Dimension name
[xyl|zltlw]units = string # x or y Dimension units

```

To create the time in as a relative time, we need the units we want to use, and a time value.

Lets get the units from the time variable above

```
t_units = t.units # get time 't' units  
print t_units  
months since 1978-12
```

Now let's take the first value:

```
t0 = tim[0]  
print t0  
1.0
```

Now we can create the relative time:

```
time_rel = cdtime.reltim(t0, t_units) # t0 only  
rank 0  
array (scalar only!!)  
print time_rel  
1.00 months since 1978-12
```

To convert it to a component time, use:

```
time_comp = time_rel.tocomp() # convert  
relative time to  
component time  
print time_comp  
1979-1-1 0:0:0.0
```

You can learn more about other ways of querying the files, variables, dimensions and axis by scanning through the [Quick Reference Guide to CDMS](#) or by following the [CDMS Basics](#) tutorial.

To learn more about the time manipulation, see the cdtime module description in the [Chapter 3 of the CDMS Manual](#)

```
continents = 0,1,2,3,4,5,6,7,8,9,10,11 # if >=1, plot  
continents  
# 0 signifies "No Continents"  
# 1 signifies "Fine Continents"  
# 2 signifies "Coarse Continents"  
# 3 signifies "United States"  
# 4 signifies "Political Borders"  
# 5 signifies "Rivers"
```

See [VCS Manual, Chapter 4](#) for extensive list of plot keywords.

Lets use plot command keywords to override some of the plot attributes:

```
v.plot(data,t_asd, comment1='Comment1',  
comment2 = 'comment2', comment3 = 'comment3',  
  
comment4='comment4',file_comment='file_comment',  
long_name='long_name',name='name',units='units',  
  
xname='xname' )  
.
```